Unsupervised Zero-Shot Anomaly Detection in Wind Turbines using Variational Autoencoders

David Geddam
Bob Jones University
dgedd236@students.bju.edu

Abstract

Background: In today's generation, wind energy faces significant operational challenges like turbine failures, due to unpredictable weather moments, harsh environments, etc, where a single mechanical fault can lead to significant damage to the turbine, with sometimes little to no prior warning. Failures like this, result in massive repair costs, and could have a significant impact on Wind Farms. Traditional systems for monitoring these assets, in today's world use supervised models. This approach is powerful and effective, but they do require vast amounts of labeled failure data to know in every way, how a turbine has broken in the past. In today's world, where obtaining this kind of data is often scarce because companies work hard to prevent these failures from happening. This creates a problem where the model cannot predict a breakdown it has memorized or has not been specifically trained to recognize. This project addresses the need for an unsupervised anomaly detection system capable of identifying mechanical drift before it gradually escalates to system-wide failures. Instead of teaching the model what a "broken" turbine looks like, we focus on entirely what the context and "physics" of normal operations are. I take advantage of a fundamental principle, specifically Cube's law ($P \propto v^3$) [2]. This law tells us that a turbine's power output is proportional to the cube of the wind's speed. The model learns to map the continuous latent space of healthy operations, creating a "blueprint" of the healthy turbine. To give VAE enough context, we transformed approximately 100 raw sensors into more than 4,755 features by calculating rolling means and standard deviations of each single row. These features had captured short term "effects" and "trends". We had to later drop a rolling window of a 24 hour step window due to Low VRAM challenges. By compressing these thousands of features into a latent space, the model is forced to find the "master" variables, such as learning relationships between wind power and speed, rather than just memorizing random noise. By framing this as a Zero-shot out of distribution task, the system catches new problems and strange physical patterns that shift away from the blueprint. This allows us to catch strange patterns that precede a mechanical breakdown, thereby enabling predictive maintenance, and avoiding large costs to repair these systems.

Methodology: The process began with selecting the data of three turbines (Assets 12, 15, and 16) from a number of turbine assets (1-90) in a specific Wind Farm [3], which were later verified to be clean of any known failures, by comparing the chosen assets against eventinfo.csv where the CSV had the data for when anomalies were recorded. This specific pipeline was designed to trigger any critical errors and stop if any training asset was found on the failure list.

To give the model enough context to see the "trends" rather than just single types of information, we transformed roughly 100 raw sensor readings into a massive set of 4,755 features, by calculating rolling means and standard deviations.

- **Rolling Windows:** We have used two windows for the model to have enough context; a one-hour window (6 samples), to catch sudden mechanical drifts, and a twenty-four hour window (144 samples) for capturing long-time effects.
- **Memory Optimization:** Due to significant challenges, like the one we had for the 30GB RAM bottleneck, we optimized the pipeline by removing the 24-hour window, lowering the threshold from 3-σ to 2-σ to increase sensitivity, and casting data to float32 instead of float64, bringing our features down to over 2,700 features, thereby reducing the memory footprint by 50%, allowing our project to run on standard T4 GPU hardware without crashing.

Further on to the process, deep learning models require raw tensors instead of raw CSV dataframes, so we converted the dataframes while doing cleaning. First, We dropped the sensor data that had zero variance to prevent NaN errors, and second, we applied standard scaling.

- **Formula:** The scaling process centers the data at zero and scales them by standard deviation:

  $$z = \frac{x - \mu}{\sigma}$$

  So what does standard scaling do? It prevents big numbers (e.g., Power levels) from dominating small numbers (e.g., Vibration levels). This is critical because it ensures that a small vibration signal (0.001) isn't ignored just because a power sensor is huge, like a value of 1000. Firstly, it subtracts the mean: This slides the whole graph left or right, so it is centered at zero. Then, divided by standard deviation: This shrinks or stretches the graph so it isn't too tall or too short.
- **Splitting:** Data was split into 80% for training and 20% for validation.

The core of our detection system is a Variational Autoencoder (VAE). This is the pipeline that followed:

- **Compression:** The encoder shrinks over 2,700 features through a 256-node hidden layer into a 10-dimensional latent space. This process forces the model to ignore noise and find the "important" variables, such as the relationship in which the power output of turbine is proportional to the cube of the wind speed ($P \propto v^3$).
- **Reparameterization Trick:** We utilized a trick to make the model efficient, we used an archer analogy to simply understand this; instead of hitting a single point, the model learns to aim for an area. We have used the following formula:

  $$z = \mu + \sigma \odot \varepsilon$$

  where z represents the latent vector (arrow position, where it lands on the target), μ represents the mean (aim, where the archer is trying to hit), σ represents standard deviation (looseness, how much looseness in the bowstring), $\odot$ represents element-wise product (application, the way wind physically pushes the arrow), and ε representing the random noise (wind, unpredictable outside forces).

The training was conducted for up to 200 epochs using a dual-objective loss function.

- **Loss Functions:** I used a combined loss function that balances two competing aims:

  $$\mathscr{L}_{\text{total}} = \text{MSE} + \beta \cdot \text{KLD}$$

  where,

  $$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2$$

  where x_i represents the original sensor data, $\hat{x}_i$ represents the model's attempt to recreate the sensor data. Here, mean square error (MSE) focuses on the model's accuracy. It forces the decoder to reconstruct the original input.

  $$\text{KLD} = -\frac{1}{2} \sum (1 + \ln(\sigma^2) - \mu^2 - \sigma^2)$$

  While MSE focuses on accuracy, Kullback-Leibler Divergence (KLD) [1] focuses to organize the latent space into a better, predictable bell curve to distribute its findings.

  By forcing everything into a bell curve, there is no isolation: Every point is surrounded by other valid points, and interpolation: The model is forced to make the transition from one point to another smooth.

Note: If we try to predict variance or stddev instead of log variance, the output maybe negative, so by predicting ln(var), the output can be negative, -5, $e^{-}5$ is a positive number

- **Early Stopping Logic:** Also, to prevent overfitting, we used a patience counter of 10. The system took a "snapshot" of the model's behavioral state whenever it hit a new best validation score using copy.deepcopy, allowing us to go back to the most accurate version just in case if the model began to over-train in some cases. In a simpler way, if the average loss is improving and good than the previous best validation loss, save the best val loss to the lowest average loss, reset the patience counter every time that occurs, and save the model to the best model state, else, add 1 every time it does not improve, and break if reached the final patience state (10).

In the final stage, we ran inference on Asset 50, which we previously verified, had contained failures, and this asset was never seen by the model during training. The model calculated the reconstruction error (MSE) for every row of data; if the model's "healthy blueprint" could not recreate the original input, it indicated strange physical deviations from the norm. We also set a threshold at Mean + 2 Standard deviations (which we got as 1.7279 by calculation):

$$T = \mu_{err} + 2\sigma_{err}$$

Here, T represents the threshold (1.7279), μ_{err} represents the mean error, and σ_{err} represents the standard deviation. By setting this threshold at two standard deviations, i am stating that any error in the top 2.5% of the distribution is an anomaly. This was able to automatically flag any anomalies, for the asset 50 turbine, for which i tested.

Results & Conclusion: This implementation of an unsupervised VAE approach shows the ability that it can capture the "physics" of the turbine dynamics and behavior without the need for labeled failure data. By compressing high-dimensional sensor data into a very low number-feature latent space, the model learned to identify the mechanical relationships necessary for stability.

My results on Asset 50 displayed clear spikes in reconstruction error that exceeded the threshold. These spikes represent "Out-Of-Distribution" behavior, where the turbine's physical patterns began to drift away from the healthy blueprint.

In conclusion, this project provides a decent framework for predictive maintenance. We have created a system capable of identifying mechanical drifts before they escalate into catastrophic failures and high repair costs, for the damages. This approach not only reduces the chances of high repair costs, but also ensures the long-term reliability of wind energy infrastructure.

Code available on GitHub: `https://github.com/dave21-py/Predictive-maintenance-project`

Keywords: Unsupervised Anomaly Detection, Variational Auto Encoders.

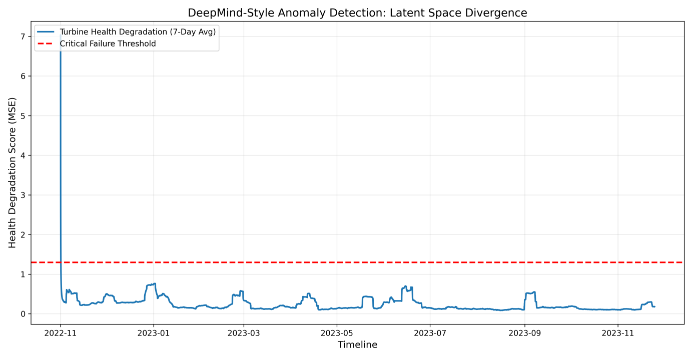

Figure 1: 7-day rolling average of the reconstruction error shows a clear 60-day degradation trend leading up to failure event, red dashed line represents 3-σ critical threshold derived from the healthy baseline.

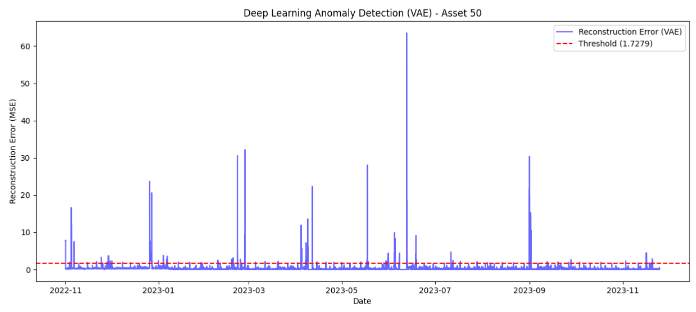

Figure 2: High frequency anomaly detection for asset 50, each blue spikes representing the reconstructing error, frequently violating the 2-σ threshold.

References

[1] Cui, J., Zhu, B., Xu, Q., Tian, Z., Qi, X., Yu, B., Zhang, H., Hong, R., (2025). *Generalized Kullback-Leibler Divergence Loss*, arXiv:2503.08038.

[2] Wikipedia. (2026). *Square–cube law*. Retrieved January 22, 2026, from `https://en.wikipedia.org/wiki/SquareâĂŞcube_law`.

[3] Gück, C., Roelofs, C. M. A., & Faulstich, S., (2024). *CARE to Compare: A Real-World Benchmark Dataset for Early Fault Detection in Wind Turbine Data*, arXiv:2404.10320.